Vivek Poojari , Satyam Rai
2019130050
TE COMPS
AIML Lab

Experiment 2

Aim: To implement intelligent agent in python that uses BFS and DFS.

PEAS:

Performance measure	Short Journey, Optimum Path
Environment	City
Actuators	Steering
Sensors	Camera

Theory:

BFS -

Breadth-first search (**BFS**) is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. Extra memory, usually a queue, is needed to keep track of the child nodes that were encountered but not yet explored.

DFS -

Depth-first search (**DFS**) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

Code:

BFS:

```
from pyamaze import
maze,agent,COLOR,textLabel
                            def BFS(m):
                                start=(m.rows,m.cols)
                                frontier=[start]
                                explored=[start]
```

```
    bfsPath={}
    while len(frontier)>0:
        currCell=frontier.pop(0)
        if currCell==(1,1):
            break
        for d in 'ESNW':
            if m.maze_map[currCell][d]==True:
                if d=='E':
                    childCell=(currCell[0],currCell[1]+1)
                elif d=='W':
                    childCell=(currCell[0],currCell[1]-1)
                elif d=='N':
                    childCell=(currCell[0]-1,currCell[1])
                elif d=='S':
                    childCell=(currCell[0]+1,currCell[1])
                if childCell in explored:
                    continue
                frontier.append(childCell)
                explored.append(childCell)
                bfsPath[childCell]=currCell
    fwdPath={}
    cell=(1,1)
    while cell!=start:
        fwdPath[bfsPath[cell]]=cell
        cell=bfsPath[cell]
    return fwdPath
if __name__=='__main__':
    m=maze(5,7)
    m.CreateMaze(loopPercent=40)
    path=BFS(m)
    a=agent(m,footprints=True,filled=True)
    m.tracePath({a:path})
    l=textLabel(m,'Length of Shortest Path',len(path)+1)
    m.run()
```

DFS:

```
from pyamaze import maze,agent,COLOR

def DFS(m):
    start=(m.rows,m.cols)
    explored=[start]
```

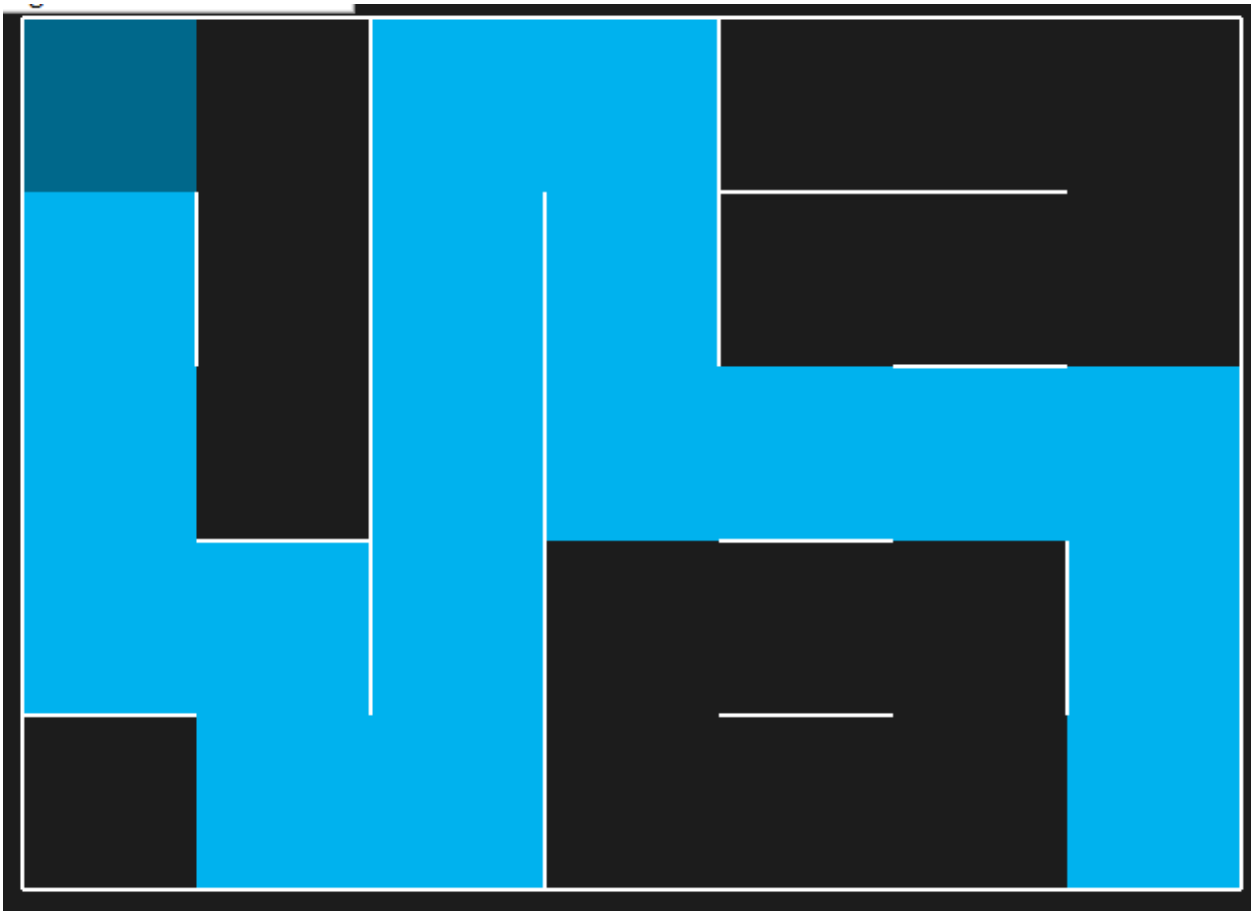
```
frontier=[start]
dfsPath={}
while len(frontier)>0:
    currCell=frontier.pop()
    if currCell==(1,1):
        break
    for d in 'ESNW':
        if m.maze_map[currCell][d]==True:
            if d=='E':
                childCell=(currCell[0],currCell[1]+1)
            elif d=='W':
                childCell=(currCell[0],currCell[1]-1)
            elif d=='S':
                childCell=(currCell[0]+1,currCell[1])
            elif d=='N':
                childCell=(currCell[0]-1,currCell[1])
            if childCell in explored:
                continue
            explored.append(childCell)
            frontier.append(childCell)
            dfsPath[childCell]=currCell
fwdPath={}
cell=(1,1)
while cell!=start:
```

```
        fwdPath[dfsPath[cell]]=cell
        cell=dfsPath[cell]
    return fwdPath



if __name__=='__main__':
    m=maze(15,10)
    m.CreateMaze(loopPercent=100)
    path=DFS(m)
    a=agent(m,footprints=True)
    m.tracePath({a:path})
    m.run()
```

Output:



Conclusion:

The environment for this experiment consisted of a city represented by a m*m grid.
The initial position of the taxi is in the bottom right square when it just starts. Considering BFS approach, the agent will select all the neighboring squares which are empty. This process is done recursively until the destination square is reached(Top left).
If we consider DFS approach the agent will select the first empty neighboring cell and recursively continues the process until it finds the destination square. This environment is restricted only for orthogonal grids of lanes and non-curvy roads.